# Topics Toward Automated Multiobjective HPC System Management

Daniel Ellsworth

March 1, 2016

# Contents

# Chapter 1

# Introduction

Large-scale computation, involving computer systems made of thousands of networked compute nodes, has had an undeniable impact on how we live. The influences of the Internet and cloud services are pervasive in modern American society. Big science is dependent on cluster computing to gather and analyze experimental data as well as simulate phenomena. Marketing and national security depend on large scale computing to monitor, analyze, and correlate human behavior.

Extremely large computation platforms come with extremely high operational costs. Private individuals do not experience the cost of Internet connectivity or the services used, as the costs are amortized over all users. The vast majority of the Internet functionality is housed in massive data centers distributed all over the world and connected by bundles of wired and wireless links, any of which may span thousands of miles. Initial procurement of facilities, hardware, and network links carries substantial cost and must be done prior to activating a system. Network links and computing hardware must be continuously maintained and power costs are incurred on an ongoing basis. Owning organizations directly experience these capital and operation expenses and must allocate their finite financial resources to cover them.

High Performance Computing (HPC) systems are a special class of large scale computing platform since HPC systems are designed for completing extremely large computations and tend to be single-site single-owner systems. Comparison of HPC systems is typically done by evaluating their rate of computation, measured in Floating Point Operations per Second (FLOPS). Current top systems operate at 10s of petaflops[1]. The power consumed to reach these speeds is substantional, with some systems exceeding 10 megawatts. A common estimate, as of the time of this

[1]One petaflop is $10^{15}$ flops.

| Attribute | 2010 | 2018 | Increase |
|---|---|---|---|
| System Peak | 2 Pflops | 1 Eflop | $\mathcal{O}(1000)$ |
| Power | 6 MW | 20 MW | $\mathcal{O}(10)$ |
| Memory | 0.3 PB | 32-64 PB | $\mathcal{O}(100)$ |
| Node Performance | 125 GF | 1-10 TF | $\mathcal{O}(10) - \mathcal{O}(100)$ |
| Node Memory BW | 25 GB/s | .4-4 TB/s | $\mathcal{O}(100)$ |
| Node Concurrency | 12 | 1-10k | $\mathcal{O}(100) - \mathcal{O}(1000)$ |
| Total Concurrency | 225 000 | $10^9$ | $\mathcal{O}(10000)$ |
| Total Node Interconnect BW | 1.5 GB/s | 200 GB/s | $\mathcal{O}(100)$ |
| MTTI | days | 1 day | $-\mathcal{O}(10)$ |

Table 1.1: Table showing the estimated properties of first generation exascale systems compared to 2010 petascale systems.

writting, for machine energy costs is roughly one million USD per year per megawatt, indicating that some top systems cost over 10 million USD per year to power[2].

Higher-level organizational objectives ultimately drive spending, not merely technical goals. For commercial organizations, these priorities tend to be driven heavily by profitability since profit is required for continued existence. For governmental organizations, these priorities tend to be driven by the organizational mission, providing the justification for that entity's funding. In either case, the computing systems are subservient to and require justification based on their ability to support these high-level priorities. Deployment of a technical solution introduces additional risk management concerns to the organization, since theft, loss, or other disruptions may threaten the organization's overall ability to operate. Given that organizational funding is limited, a computing solution may be too expensive for adoption, regardless of technical merit.

The US Department of Energy (DoE) is a major consumer of HPC systems. These systems are used to support a national security mission via advanced applied and fundamental science activities. Current DoE exascale initiatives are driven by a desire to support more and larger science efforts. Table 1.1 is a familiar table for HPC researchers[3] regarding the challenges moving from 2010 petascale systems to a 2018 exascale target. Many of these challenges appear similar to scaling in industry and are potentially addressable via the horizontal scaling strategies in industry:

[2]As of November 2015, the Tianhe-2 is listed as the world's fastest computer according to the Top500 list [1]. The system is reported to consume nearly 18 MW. RIKEN, in fourth place on the Top500 list, is reported to consume over 12 MW.

[3]The table's origin is sometimes attributed to Pete Beckman.

concurrency, fault remediation, and interconnection bandwidth. One major and confounding new design challenge for HPC is an administrative, more than technical, restriction that the computer have a finite power budget that grows significantly slower than the compute performance.

Current HPC systems are power over-provisioned, meaning that power infrastructure and procurement are sufficient to safely run the system at peak theoretical power consumption. One proposed strategy to achieve exascale is to build a hardware over-provisioned system, meaning that there is more hardware available than can be simultaneously powered. Hardware over-provisioning introduces a new organizational risk since a full computational load would induce an unsupported rate of energy consumption, potentially damaging the computer and power infrastructure. To address the new risk, new power aware resource management systems will be needed to guarantee total power consumption remains safely within bound, protecting the system from damage, while simultaneously optimizing performance to align with other organizational priorities.

There is a practical tension between what can be done and what an organization can afford to do. New technical solutions will likely be needed to operate future systems in alignment with management constraints and priorities. In the next three chapters we will look at three Computer Science (CS) topic areas that are likely relevant for the design and implementation of these solutions.

1. Information Flow Processing (IFP) - Research that could be used for collecting information needed to manage a large scale-computer system.

2. Scheduling - Research into matching work and resources.

3. Energy/Power Aware Computing - Research looking at a major operational cost for HPC computing platforms.

# Chapter 2

# Information Flows

Effective system management requires visibility into the system state. A continuous flow of information regarding the system's current state is required for management to detect issues needing remediation and inform future decisions. Simple questions in an High Performance Computing (HPC) system , such as "have any nodes failed?", requires collection of some data indicating liveness across all nodes. More complex questions, such as "Are the failed nodes due to node level or rack level faults?", require both collection and correlation of data across many nodes. Ideally information regarding system state is rapidly available to management. Continuing with the node failure example, if events are received indicating a node will soon fail then management could proactively reassign the work and avoid interuption. Information Flow Processing (IFP) helps to address the visibility challenge by providing capabilities to gather data from a large number of sources and perform some processing as the data moves to interested sinks.

Cugola and Margara [13] unify Complex Event Platform (CEP) and Datastream Management System (DSMS) research under the unified topic of IFP. An IFP system, according to Cugola and Margara has a number of distributed sources from which information is processed as the data moves towards a sink[1]. Stonebraker, Centintemel, and Zdonik [60] enumerate 8 desirable properties for IFP systems motivated by latency and usability. The data flow or flows are typically processed as the data moves through an overlay network to address challenges associated with data volume or timeliness constraints. Hirzel et al. [25] enumerate twelve transformations on the flow processing graph that may be useful in performance optimization.

[1]A single sink seems overly restrictive, and may be broadened to include systems where there are multiple sinks.

Researchers working on IFP often dismiss traditional relational database management system (RDBMS) solutions, including active databases, as being too slow [13, 60]. The position is unfortunate since modern RDBMS solutions, with insert triggers and asynchronous queues, are capable of behavior like a centralized DSMS. An IFP system built on Oracle's RDBMS will be briefly discussed in section 2.4. The program analysis community, until recently, has been similarly dismissive of using traditional databases in program analysis due to a similar belief in poor performance [2]. While soundly outside of the scope of this research, there is reason to believe properly configured RDBMS's may be suitable for IFP applications.

## 2.1 Simple Event Processing

A simple example of IFP is a pure publish/subscribe system. The only processing provided by the system is a null transform on the data. Generally, publish/subscribe systems do provide filtering on event fields. Differences in the structure of publish/subscribe overlay, filtering strategies, and routing strategies lead to differences performance and service guarantees.

Hermes [54] provides a distributed publish/subscribe system using an event forwarding tree. Events are typed, defining the fields expected in events, and a node is elected as the root for each event type. Event publishers and subscribers are arranged in a tree shaped broker overlay per type. Publication advertisements and subscription filters propagate toward the root. Published events propegate toward the root and each intermediate broker with a matching filter propagates the event along the reverse path of the subscription.

CIFTS [23] describes a publish/subscribe system targeting log like data in HPC systems. Subscribing clients are connected to a static tree of brokers. Publishers send events to brokers asynchronously, which are delivered to each subscribers' queue. Each subscriber is responsible for polling their queue to detect and read incoming events. In addition to filters based on event contents, CIFTS supports event namespaces, which effectively define named event streams.

[2]In the 1980s Linton [41] used a relational database as part of an analysis tool and commented on the advantages of an RDBMS for reasoning and structuring his solution as well as the performance challenge. Databases have been effectively absent from the program analysis literature until recently where datalog has been used for efficient points-to analysis [37, 59, 4].

## 2.2 Complex Event Platforms

CEP systems typically target environments where there are a large number of atomic event sources and analysis of event patterns is required to indicate that some action should be taken. A publish/subscribe style interface for event producers and consumers is generally provided by the system. Event sources publish events to the CEP and event consumers subscribe to the CEP for events of interest. Usually the internal structure of events are unspecified or underspecified by the platform to allow for flexibility. Mechanisms are also provided within CEPs to describe how to generate "complex events", events composed of other events, based on the patterns in the event stream.

Carzaniga, Rosenblum, and Wolf [10] present a distributed CEP platform, SIENA, supporting efficient untyped event notification. Sources and sinks interface via a broker network that handles event filtering and pattern detection, based on event content, as events flow through the overlay. Prior to publishing an event, an event source must advertise a filter that matches all events that might be published. An event sink will subscribe to a broker with a pattern describing the events the sink is interested in. Coversets, encapsulating advertisements and subscriptions, are propagated between brokers to generate routing information for efficient event forwarding. Follow-on work improves routing performance by reducing the number of times filter conditions must be evaluated to make the routing decision[11] More complex predicates, such as 2D spatial filtering have also been added to SIENA [35].

PADRES [40, 30] provides an IFP platform to efficiently generate composite events. Subscription and event routing are managed via rules distributed through the broker overlay. A rule describes the relation between fields of one or more events, using logical AND or OR operators to chain together multiple constraints. Additional PADRES features include an SQL-like query language for subscription and persistent event stores to support new subscriptions involving historic data.

Apache Storm [42] uses a data flow abstraction for data processing that can be adapted to provide CEP like services. Computations are defined using a directed acyclic graph of operator elements, referred to as "bolts". Events enter Storm via spouts, traverses the bolts in the data flow graph, and exit the system as new events. Bolts are arbitrary code and can perform any operation on the data flowing through the bolt.

Over the past decade a team at Georgia Tech has been working on a framework, named EVPath, designed to facilitate rapid construction of CEP overlays for

HPC [18] and data center domains. ECho [17] uses EVPath for construction of a publish/subscribe system for HPC systems based on typed events and event channels. Monalytics [36, 66] uses EVPath to implement a realtime datacenter monitoring platform. Flexpath [14] supports the construction of scientific workflows by interconnection of scientific codes using ADIOS [43] for IO via an IFP overlay. ADIOS with Flexpath treats data writes as events and uses the overlay to route write events to the correct readers.

## 2.3 Streaming Databases

Datastream Management Systems (DSMSs) typically target environments where there are a number of tuple streams that must be correlated in realtime to support some action. A query language is usually provided to allow sinks to express the data of interest and the DSMS then encodes the query into the overlay. The queries can be thought of as running continuously, as each new record received by the DSMS flows through the processing overlay and potentially causes a new result to be delivered to the sink. Records typically have a well defined schema and each stream represents one type of record.

Arasu, Babu, and Widom [5] propose and implement a continuous query language, CQL, for interacting with DSMSs. CQL is based on the relational algebra constructs of SQL and also supports mixing tuple streams with traditional relational tables. Streams in CQL are conceptually mapped to tables using windows, a window is a set of tuples over some interval within the stream. Referencing a stream within a query involves providing the stream's name and a window specification. Derived streams, streams resulting from CQL queries, may be referenced in other CQL queries and must provide a stream type that informs which tuples are returned.

Aurora [12] uses a conceptual model involving boxes and arrows, which respectively contain transformation operations and connections between boxes, to implement continuous processing of information flows. Data flowing through Aurora is represented as tuples and the processing graph must be loop-free. Load balancing is achieved by migrating boxes between compute resources and potentially mutating the processing graph with stream splits and joins. To support failure recovery in distributed environments, lazy deletion of tuples allows for retransmission and recomputation when a failed box is reinstantiated. Aurora's authors also mention quality of service (QoS) concerns when designing the load shedding and latency estimation features.

SASE [67] considers the flow processing problem as execution of table based queries over temporal windows. Queries in SASE express tuple patterns of interest and executing a SASE query returns complex events composed of the tuples generating the match. An interesting feature of the SASE event language is the inclusion of event negation, querying for patterns involving the nonexistence of particular events. SASE uses a query planner to generate the necessary predicates and state machines for phases of the processing pipeline. SASE's authors cite the assumption of global total temporal order and the inability to use the output of a SASE query as a tuple source as limitations of their design.

Cayuga [16] considers the flow processing problem as being similar to string processing by finite automata. Cayuga Event Language (CEL) is an SQL-like language used in Cayuga to describe, based on some set of named tuple streams, a desired output tuple stream. CEL queries are compiled into automata by the Cayuga engine. At runtime Cayuga has only a single instance of each automata but keeps bookkeeping data to track conceptually concurrently executing instances of an automata. The restriction that tuples be processed in global temporal order seems like a challenge for scaling Cayuga.

## 2.4 Other Systems

Between 2005 and 2011 I worked on several commercial systems that fit in the IFP space and had committed timeliness and data volume requirements in the system service level agreements (SLAs). These systems used a three layer design and microbatching strategy for event ingestion and a traditional clustered RDBMS to present the data to consumers. Events were received or pulled from the source systems, using the source's preferred protocol, by the extractor layer. The extractor layer would timestamp and give a unique identifier to each record before passing the events to the formatter layer. The formatter layer would convert the batch of events to database rows that could be quickly ingested by the database and flag event data that did not match the expected schema. The load layer would add the formatter output to the database tablespace, update partitions, and start any postprocessing processes required for the newly ingested data. The 2005 production deployment was scaled to ingest 2.5 TB of data per day at an insertion rate of over 30k events per second with a maximum 40 second SLA on time from extraction to availability in end user reports. Internal experiments conducted with the 2005 deployment reported near linear weak scaling for the tested node counts and data volumes with response times well under the SLA.

Cache [61] is a centralized IFP system that provides both event and RDBMS like interfaces. Cache is designed to support high performance complex event processing through the registration of automata within the Cache server. Automata in Cache are written in a C like language and may inject new events or send data, via a network socket, to the process that registered the automata. As new events are inserted into Cache, automata associated with the effected tables are triggered and run asynchronously in other threads. Each event received by Cache is stored as a record in a ring queue based on insertion time, where the specific queue is related to the event type. The ring queues can be interacted with like tables in a traditional RDBMS via SQL, with the conspicuous lack of the join operator.

Spark [69] represents big data processing as transformations from a large distributed data set. "Queries" in Spark are written as an expression of data transformations. Executing the "query" involves applying the transforms in order towards the underlying data and caching intermediate results as the data flows towards the requester. Caching and lineage allows for recomputation when faults are encountered. The streaming extensions to Spark [70] support streams as sources, in addition to files, and use microbatching based on event arrival time to support data processing over windows.

LDMS [2] has recently gotten a lot of attention in the HPC space. The monitoring system provides support for high fan-in[3] metric collection across an HPC machine using a tree shaped overlay. A plug-in model is used for extending the samplers to collect new metrics as well as extending the supported output formats. The LDMS aggregators are located on separate nodes from the monitored workloads and polling is used to retrieved data from children. A child may simultaneously participate in several overlays, with aggregator polling intervals being individually configurable. The only aggregation model discussed is effectively concatenation.

MRNet [55] also provides a tree shaped overlay for aggregation and has been used by several projects in the HPC space [55, 39, 6, 46]. Agelastos et al. [2], in the related work section, argue the superiority of LDMS for the metric collection activity due to a better alignment of LDMS features (compared to MRNet's features) in LDMS's deployment context.

[3]High fan-in refers to enviroments where there are a very large number of sources all sending to one sink.

## 2.5 Other Concerns

Event generation is assumed to be asynchronous across sources. In environments where system clocks are well maintained and the time tolerances are wide enough[4], source generated timestamps can be used to provide a global temporal partial order on events. Latencies in event propagation add an additional challenge for pattern matching involving event order since events may not be received in the globally agreed temporal order. Correct ways to address temporal issues are unclear due to potential impacts on the competing design goals of correctness and responsiveness.

Deployment of a single IFP platform across many users may introduce security concerns since not all events should be directly shared with all users. CS-DSMS [68] targets shared data centers where conflicts of interest exist between users. CS-DSMS event streams carry a vector describing the security level of the data within the event stream. The security vector of the querying process must dominate a stream's security level to include the stream's data in a computation. Minami et al. [44] describe a publish/subscribe system, with event aggregation support, where a publisher defines an access control list (ACL) to control which nodes may subscribe directly to or receive aggregates based on the publisher's events.

Direct performance comparison of solutions built on general purpose IFP system to specialized solutions may be unfair. For a single application with tightly coupled data collection, using highly optimized communication primitives like those provided by MPI [62, 26] will almost assuredly be more performant than using a more general flow processing solution. IFP platforms allow collected data to be reused across multiple consumers, potentially amortizing the collection and transport costs across many concurrently executing applications. The value proposition for IFP within a solution's design must consider a broader view of implementation context than the specific solution being built.

## 2.6 Takeaways

In context, system wide management problems are multi-dimensional and crosscutting. IFP systems provide a way to flexibly collect and distribute information across application boundaries and distributed execution environments. The performance information useful for resource management within a system can

[4]Technologies such as NTP are able to support clock synchronization on the order of 10s of milliseconds and other techniques can reach synchronization within microseconds [31]

be thought of as an event stream with events generated at the sampling interval for monitored resources. Systems like PTRADE [28], if extended to observe and optimize machine wide performance, would likely benefit from having realtime visibility into configuration effects across the machine. Systems like Argo [53] could use an IFP-like information service to avoid multiple collection of metrics in the presence of multiple resource managers, running in parallel or hierarchically.

Intuitively, distributed IFP solutions feel unavoidable at scale. Increasing data volumes should overwhelm centralized solutions due to bandwidth and compute resource limitations, yet the complexity of distributing the processing appears in many cases to overcome the anticipated gains from distribution. Results from Ellsworth et al. [19], Koliousis and Sventek [34], and experience industry indicate centralized solution can perform reasonably well at fairly large scales. The Koliousis and Sventek [34] paper is germane since the DEBS challenge specifically targets the IFP solution space and Cache was able to outperform the other IFP platform submissions. More work is likely needed to understand how to model performance and cost to understand when and in what environments the overhead of distributed IFP can outperform more centralized processing. Additionally, a closer look at performance evaluation criteria may be warranted since an application with specialized data collection will most likely outperform solutions incurring the overhead of a generalized IFP solution.

# Chapter 3

# Scheduling

Scheduling, as a general problem area, can be described as matching work to be done with resources to accomplish that work. The association is commutative; scheduling can equivalently be described as matching resources to accomplish work with work to be done. Using either description, there will be constraints on the resources or work available that restrict the set of possible schedules. Ideally the work and resources will allow a schedule to be found that fits within the constraints. Practical scheduler implementations cannot rely on knowledge of such a schedule a priori and must be able to handle overloaded conditions; conditions in which there is insufficient resource or insufficient work.

Algorithmically, scheduling can be presented as a packing problem where the units being scheduled are the things to pack. Given some set of things to be packed and a structure with constraints to pack them, can all of the items be packed? The knapsack problem is perhaps the canonical articulation from the algorithms community. In the general case, computing decidability for the knapsack problem is NP-complete. The optimization problem, determination of an optimal packing, is NP-hard in the general case.

## 3.1 Work to Resources

A common approach to scheduling is to start from the computation work to be done, at some granularity, and assign that work to the available resources. Common granularities in High Performance Computing (HPC) literature are jobs and tasks. In other areas, processes and threads are common granularities for work scheduling. Some runtime systems suggest implementers over-decompose computations into units of work too light or too numerous for standard OS threads, such as Charm++

"charms", with the expectation that the runtime can better utilize the hardware through scheduling the smaller work units.

Handling overload, when work scheduling, is generally handled by queuing work to be done until resources are available to service the work. The problem of scheduling is thus often recast from a packing and optimization problem to a queuing problem. Queue the work to be done, then dequeue and assign work to resources as the resources become available to service the work. Specifics of how the queues are emptied determine the properties of the schedules produced.

### 3.1.1 Jobs

Jobs generally represent complete computation activities that do not depend on other concurrent computation activities. A job description or script will enumerate the resources required, the specific computations to run, and the data to use. Structured as independent and encapsulated units of work, jobs have nice properties for scheduling since they can be run whenever resources are available and the scheduler makes the assignment. Often the job scheduling problem is evaluated with respect to FIFO queuing, commonly referred to as First Come First Served (FCFS) in the job scheduling literature. A priority queue can be used instead of a FIFO queue without loss of generality and enables scheduling based on properties in addition to submission time.

Modern schedulers are parallel job schedulers as opposed to pure batch schedulers. In pure batch scheduling systems, only one job would be allowed to run at a time on the system. Parallel job scheduling allows jobs from the queue to be run concurrently if sufficient resources are available. When the highest priority job in queue cannot be scheduled due to insufficient resource availability, backfilling schedulers will scan ahead in the queue for jobs that will fit in the available resources and not overly delay the start time of the highest priority job in queue[21].

Most users of HPC systems interact via a job scheduler, such as SLURM or Torque. Important parts of the job resource description on these systems typically include the number and types of resources needed for the job, the expected runtime of the job, and the account to charge the machine time against. Estimated runtime can be used to allow the scheduler to use backfilling to reduce the time to launch while mitigating against starvation issues that backfilling introduces. Associating an account for chargeback with each job supports priority queue disciplines where the value of the job to the organization owning the machine can be included as a factor in scheduling. These job schedulers are designed for HPC job profiles, which tend to involve jobs that use a large number of nodes for long periods of time.

Sparrow [47] targets environments where jobs involve multiple nodes but each job is expected to be short, a job profile aligning with Hadoop and Spark queries. Jobs are expressed to Sparrow as a sequence of tasks to be executed by workers. Sparrow uses a set of uncoordinated Sparrow schedulers sharing a pool of workers. At schedule time, workers are sampled to determine worker workload and tasks are distributed to sampled workers with least load. Since the Sparrow schedulers are almost stateless, scheduler failure is handled by resubmitting the outstanding tasks to one of the live schedulers.

### 3.1.2 Tasks

Within a parallel job or parallel program, the work can be subdivided into tasks. Tasks are units of work that may depend on one another. The task scheduler is able to concurrently schedule any tasks where the dependencies have been met. Since the tasks that can be scheduled may evolve during runtime, task scheduler logic must often run during program execution. Overheads are incurred against execution by the time taken to make the scheduling decision and time need to move the associated work. In simple cases, tasks may be assigned to compute resources at task generation time such that each compute resource has an equal number of tasks.

A simple example of parallel tasks would be the body of a PARALLEL_FOR loop. When the "loop executes", a task is spawned for each time the loop body would be executed serially. After the tasks are spawned, the runtime can execute the tasks in parallel across the available compute units. Once all of the tasks representing the body of the loop have completed, the task containing the instructions following the loop can be scheduled for execution.

In addition to scheduler overheads, task scheduling carries challenges due to load imbalance. A seemly natural approach to task scheduling given $c$ compute units and $t$ tasks, where $t \gg c$, an equal number of tasks could be assigned to each compute resource. At a practical level, even in the presence of tasks with equal compute cost, data access costs can lead to significant performance variation depending on which compute resource a task is assigned to. In the general case the cost of a task cannot be computed a priori since this would effectively require solving the halting problem. Assigning an equal number of tasks to each compute resource can result in imbalanced load if some resources become idle (unloaded) while other resources still have tasks in queue.

Work stealing is a task scheduling approach in which unloaded compute resources take tasks from queues located on other more loaded compute resources. In a centralized formulation, there is a global work queue containing all of the tasks to be performed. When a compute resource has capacity, the compute resource requests work from the central work queue. In a diffuse formulation, the centralized work queue is replaced by the work queues of some number of neighbors. For either formulation, a compute resource may request additional work in advance to hide the latency involved in migrating the task.

The sandpile scheduler [38] is similar to work stealing but is proactive on task reallocation. In sandpile scheduling, task queues are arranged as an undirected graph, which the authors suggests should be a small world graph. Each scheduling cycle, the queue depths are exchanged by neighboring nodes and tasks "avalanche" from deep queues toward shallower queues. The sandpile approach is diffuse and the authors indicate rapid convergence to a near optimal schedule for their experimental test sets.

Zheng et al. [71] investigates a hierarchical approach to load balancing for Charm++. Load balancers are arranged in a rooted tree, where all nodes are balancers and leaves are tasks. Tasks are represented by tokens, held by the parent load balancers, allowing the load balancing computation to be done without directly operating on the tasks. At the end of a scheduling cycle a global collective is used to request work from the formerly owning leaf scheduler to scheduler currently holding the token for the task. Reported experimental results indicate that the hierarchical approach can produce application load balances similar to a centralized approach with less wall clock time lost to overhead.

Pearce has done considerable work in the space of load balancing for HPC codes. These codes involve simulation of physical phenomena and tend to have spacial geometry. In simulations where tasks represent a spacial volume within a mesh, the amount of work associated with a cell may change over time as the simulation progresses[1]. The scheduling challenge involves distributing imbalanced tasks across the compute resources so that all resources have approximately the same amount of work or modifying tasks such that all tasks contain approximately

[1]We may consider the case of simulating ping-pong balls bouncing within a box. The box is divided into a mesh of cells and a task encapsulates the simulation work to be done within a particular cell. As the ping-pong balls move through the simulated volume, the balls move between mesh cells. Mesh cells with a higher density of ping-pong balls will require more compute resources since there are more interactions to compute.

the same amount of work.

Pearce et al. [51] suggest traditional load imbalance metrics provide little utility for selecting a balancing algorithm. The traditional metric for load imbalance considers the ratio of the most loaded component to the average load, $\lambda = \frac{L_{max}}{L_{avg}}$. Statistical moments might also be considered for load imbalance metrics but Pearce argues statistical moments provide little insight to guide balancing algorithm selection. Using an element-aware load model, where compute load and work element costs are reported, the performance of the load balancing algorithms can be estimated by the scheduler. Results from the work indicate the model selects the best algorithm in almost all cases.

Pearce et al. [50] investigate load balancing in N-body simulations using an approach built on hypergraph cuts, motivated by the insight that the compute work is related to the number of interactions rather than the number of particles within the task. An N-body simulation computes the state of a physical system based on the pairwise interactions between bodies within the simulation (e.g. location of the planets in a solar system based on gravitational interaction). Bodies can be thought of as vertices within a graph and interactions as edges within the graph. In the hypergraph, weighted vertices [2] represent adjacent edges from the initial graph (e.g. interactions that share a body) and edges represent both the body and the body's interaction edges. The scheduler partitions the hypergraph, with the objective of having an equal vertex weight in all partitions and a minimal number of cuts, then assigns a partition to each worker process. Adaptive sampling is used to substantially reduce the size of the hypergraphs constructed and the authors report a significant performance improvement using their approach.

In Decoupling Load Balancing [52], Pearce et al. suggest decoupling scheduling activity from simulation activity. Classically, a simulation pauses after a number of iterations to execute its load balancing computation on the same resources being used to compute the simulation. In the proposed decoupling, processing external to the simulation is used to asynchronously compute a new work distribution based on information periodically sent by the simulation. An additional benefit of the decoupling is the ability to scale the scheduler and simulation compute resources separately. New assignments are determined by the scheduler and are applied lazily by the simulation. The initial experimental results use simulation runs with 65k compute processes and achieve a load imbalance of under 5% while reducing the

[2] The paper was not clear on how the vertex weight is computed. I would guess the weight is related to the number of edges the vertex represents in the initial graph.

scheduler overhead by a factor of 15.

## 3.2 Resources to Work

The schedulers had control over when and where work was done in the previous section. The resources were fixed and the work was distributed across the resources. The scheduling problem considered in this section involves contexts in which when and where the work occurs is outside of the scheduler's control. The work to be done concurrently is fixed and the resources are distributed across the work. While these both appear to be sides of the same coin, the inability to directly delay work makes queuing strategies less applicable.

Allocating RF spectrum appears to be a resource scheduling problem. The bandwidth of the underlying medium is the resource to be scheduled and is partitioned by frequency. The scheduler is allowed to assign frequencies to transmitters. Work is not controlled by the scheduler, when and who transmits is determined by the transmission source. Only a single transmitter by use the resource at a time, concurrent use without additional protocols would results in garbled transmissions.

Circuit switched networks provide another example of resource scheduling. A simple circuit switched network contains a finite number of links (resource) and each link may participate in only one circuit at a time. The network has little control over the work done by an end point or when and which end points will communicate. During the setup phase of the connection, the network reserves (schedules) links for use by the requesting end point. If insufficient resources are available, the network must reject the connection attempt but the work bears the responsibility of choosing to fail or retry the communication.

Store and forward packet switched networks, like IP networks, do not fit this model for resource scheduling. Conceptually, each router in a network is a node in the network with a number of interfaces/links (resources). Each packet received by the router is a work unit that must be scheduled on a link. The packets (work) can be held in queue until the resource becomes available to assign the work to; violating the constraint that the scheduler has little control over when the work executes. quality of service (QoS) and Software Defined Networking (SDN) features erode this objection as both technologies enable packet switched network configurations that share some resource reservation qualities found in circuit switch networks.

In Hedera [3], a central scheduler assigns physical network paths to logical network flows and can be implemented using SDN. An optimal schedule is one in which no flow is network limited, however Hedera does not attempt to schedule the bandwidth associated with the flows. Underlying packet queuing and retry mechanisms within the network handle the case of overload Equal-Cost Multi-Path forwarding (ECMP) is used as the performance baseline for comparison, the ECMP strategy has each router independently select from multiple outgoing interfaces based on a hash computed from the packet header. Evaluated Hedera strategies include a greedy first-fit strategy and a simulated annealing strategy. The work is evaluated experimentally and in simulation, showing improvement in resource utilization.

Sharma et al. [58] use QoS mechanisms to support bandwidth reservations for bulk data transfers across networks. Reservation requests indicate the amount of data to transmit, when the data will be available for transmission, and a deadline for when transmission must complete. The domain controllers coordinate across network boundaries to schedule bandwidth for the transmission, which the local request schedulers instantiate via network QoS configuration. Cases where the network may provide bandwidth exceeding the disk bandwidth are considered and addressed by including a maximum rate in the reservation request. When a bandwidth reservation cannot be made for a reservation request, the request is rejected and no bandwidth is reserved.

In POWsched [20], a central scheduler assigns power to physical processor sockets. An optimal schedule is one in which no socket is power-bound during execution, however POWsched does not attempt to schedule work on the sockets. Overload has the side effect of reducing runtime performance since the processor must down clock to maintain the assigned power bound. Power scheduling is done using a simple closed loop control strategy, distributing power from under consuming sockets to sockets that appear power bound. When insufficient power is available, POWsched attempts to converge to a fair share power allocation across power bound sockets.

## 3.3 Interference

Most work directly on scheduling considers the entirety of the scheduled units being isolated and unshared. Using an abstraction with noninterference between units makes reasoning about behavior and structuring solutions easier. In real systems the environment is rarely so clean. Scheduling exists because the system resources are shared.

Kambadur et al. looked at performance interference between processes sharing a core with Google's production systems[33]. Their work indicates that the runtime perturbation, measured as standard deviations of instructions per cycle, of co-resident work differ significantly depending on the co-resident application. The result is, on the one hand, intuitive due to the expectation that some local resources will be contended. On the other hand, the perturbation reported is unexpectedly asymmetric and not maximized by running two instances of the same process.

Breslow et al. [9] assume that a scheduler can concurrently execute jobs from multiple users on a single node. The work is motivated by insights that coresident processes from different applications, in some cases, can improve global system performance in energy efficiency. At the node level the collocated processes compete for resources and perturb performance, which presents a challenge for the dominant chargeback model based on node compute hours. Their tool, POPPA, samples collocated job performance and allows for fairer chargebacks.

Bhatele et al. [8] investigated performance in terms of runtime for an HPC workload. Their work highlights the inconsistency of performance in the presence of a shared network substrate. The results of running the code on BG/Q architectures, in which each job has a dedicated block of the system, is extremely consistent compared to the same code run on a Cray XE6 using a Gemmi network, which is shared by all jobs.

While rarely talked about in scheduling literature, security is also a practical consideration for scheduling of shared resources. Isolation is a foundational design principle for risk mitigation and reliance on the abstraction may introduce insecurity. Colocation of two work items on a resource allows each to infer information about or perturb the operation of the other, often impacting confidentiality or availability properties. In security contexts where strict separation between users and between work units is a requirement, the amount of leakiness of the scheduling abstraction on the system should be considered in risk assessments.

## 3.4 Takeaways

Schedulers must have some mechanism to handle overload if they are to practical. Ideally a schedule exists that allows all work and resources to be satisfactorily assigned, however for many real cases no such schedule exists or is infeasable to compute. Work scheduling generally handles the overload problem using queues and often describes scheduling in terms of queue management. Resource scheduling has little recourse in the event of overload other than flat out denial of resource access or providing a fraction of the requested resource. Smooth handling of overload in resource scheduling requires the work have an ability to adapt to the assigned resources.

The majority of work surveyed on scheduling is from the HPC community and tends to be focused on work scheduling approaches. Limited work on resource scheduling within the community seems reasonable given that HPC has classically had few realtime constraints, allowing under runs to be handled by delaying the computation. Power is a fundamentally different resource to schedule since an under run of power can cause the computation to be incorrect and physically damage the hardware. Future HPC scheduling work will likely have a greater focus on realtime constraints due to current community interests in power, in-situ visualization, and in-situ analysis.

Problems and approaches from the networking community for streaming media were not included in the literature survey due to the focus towards HPC. Scheduling bandwidth resource for adaptive streaming work likely has similar properties to scheduling power resource to compute work. Like power, excess bandwidth above a threshold cannot improve stream quality and bandwidth beneath a threshold results in quality issues. Future scheduling work in HPC might be able to benefit from looking at existing work regarding media streaming.

# Chapter 4

# Power and Energy

As a by-product of their operation, computers convert electrical energy into thermal energy. A major contributor to energy consumption and heat generation within a computer is transistor switching. Each transistor converts a small amount of energy to heat when changing state, the amount is related to the voltage and switching frequency. Additionally, some energy is lost due to leak current. A model to describe the power loss due to switching is $W = \eta C V^2 f$ [15]; the watts $W$ lost are directly related to the square of the voltage $V$ and the frequency $f$ of switching. Voltage and frequency must be increased together, resulting in a nonlinear relationship between watts and instruction execution speed dominated by the $V^2$ term. Power lost to leak current is also related to the voltage [15].

Energy constrained and power constrained computing are separate but related[1] problem spaces. Energy constrained computing captures cases where the number of joules available for computation are limited. Much of the energy work in the mobile computing space is focused on how to maximize the amount of computation available before depleting the battery. Data center green computing also tends to focus on energy constraints to meet an energy bill target or reduce the energy cost associated with operating the data center. Power constrained computing captures cases where the rate at which joules can be accessed is limited. In the power constrained computing environment, the system is typically connected to the power grid (effectively unlimited energy) with the local substation and physical power distribution infrastructure limiting the rate at which power can be moved to compute components. Work on power constrained computing is not common yet, but is starting to appear in the HPC space due to the challenges involved in supporting the maximum power draw of all components.

[1]We highlight that power in watts is defined in terms of joules: $W = \frac{J}{s}$

There are two primary low level techniques for energy and power control exposed to software: dynamic power management (DPM), and dynamic voltage and frequency scaling (DVFS). Hardware that supports DPM allows software to make the processor go into low power sleep mode, greatly reducing the number of active transistors. The sleep mode results in minimal power consumption but halts computation. Hardware that supports DVFS allows software to adjust the voltage and frequency of the processor. Reducing the clock frequency may or may not impact application runtime due to the latencies involved in memory access; at a higher clock speed the processor will incur the power cost of higher voltage while executing no ops until data arrives from cache or the bus.

While DVFS results in reduced power consumption over the maximum clock speed, DVFS does not guarantee a particular power consumption is not exceeded. Work published in 2002 by Minerik, Freeh, and Kogge [45] demonstrate the ability to achieve consistent power consumption under a general purpose operating system using frequent DVFS setting changes. The authors design and describe their technique using the problem of meeting a battery life goal in a laptop. The battery life goal is achieved by using DVFS to tune for an average power consumption that aligns with the energy target. $P = \frac{E_0 - E_f}{T}$ where $E_0$ is the initial battery energy, $E_f$ is the final energy, and $T$ is the desired time to reach $E_f$ from $E_0$. The model can be extended for settings where the battery is being recharged while the power control is running.

Intel's Running Average Power Limit (RAPL) uses a technique similar to the one described by Minerik, Freeh, and Kogge [45], in addition to other proprietary techniques, to allow software to set a power limit that the processor will not exceed. Rountree et al. [56] and Inadomi et al. [29] look at the effect of RAPL across many identically configured nodes executing the same benchmark. The work shows a significant difference in runtime performance between nodes when executing the same benchmark at the exact same power bound. Differences in runtime are attributed to subtle manufacturing variations between processors. While not explicitly stated, these variations in performance encourage Ellsworth et al. to schedule power per socket rather than per job or node[20].

## 4.1 Energy Aware Computing

Energy aware and green computing are umbrella terms for techniques and technologies that reduce the energy used to complete a computation. Reducing the energy used to complete the same computation, within the time envelope, can result in lower operating costs. Much of the work in this space has a distinct optimization focus and is a point of similarity between embedded computing and HPC. The underlying problem can be stated as "given that energy has a cost, how can the energy cost be minimize and the system still complete the work?"

Battery operated realtime systems have hard operational constraints involving guaranteed maximum latency and battery longevity. Bambagini et al. [7] investigated the energy consumption properties of using DVFS and DPM and proposed a power scheduling solution for energy limited realtime systems. The schedule attempts to run the computation phases at the most energy efficient setting that meets the realtime deadline (DVFS savings) and simultaneously attempts to maximize the idle time between computation (DPM savings). A limited preemption model is used to reduce switching overheads when compared with fully preemptive scheduling and to avoiding deadline issues possible under fully nonpreemptive scheduling. Experimental results indicated that their hybrid DPM and DVFS solution outperforms DPM or DVFS only solutions.

In Adagio, Rountree et al. [57] look at reducing energy consumption of HPC applications while maintaining runtime performance using DVFS. The Adagio authors use the insight that in many cases the nodes executing a High Performance Computing (HPC) application do not arrive at MPI collectives simultaneously, causing some number of nodes to idle until all nodes have arrived. As the application executes, Adagio records the time spent between collective communication and the time spent waiting within the collective. On future iterations, Adagio adjusts the clock frequency during the computation portion of execution to reduce the time spent waiting within the collective. Adagio runtime and energy savings are favorable across a wider range of applications than the other algorithms mentioned in the Adagio paper.

Green Queue [64] also investigates energy saving for HPC applications using DVFS. The approach used by Green Queue is considerably more invasive than Adagio, involving static analysis and traced executions prior to the experimental execution. While the reported Green Queue results are impressive, with an average 17% energy savings, the results are out of context since the energy and time cost of executions needed to generate models are not considered. Failure to consider the total energy costs of the approach may challenge the viability of the work for adoption since the total cost may exceed the benefit.

Mistral [32], attempts to maintain bounded application responsiveness while saving power in virtualized environments. Virtual machine migration, virtual machine CPU fraction, and powering up/down hypervisor nodes are configuration actions available to change application power consumption characteristics. Reconfiguration of the system potentially carries a cost in time and application responsiveness which is factored in to the decision. A pregenerated table is used to estimated the cost of a reconfiguration action. Exprimental results in the Mistral paper show application responsiveness and power consumption are favorable compared to other techniques.

### 4.1.1 Computation Energy

A question one might ask is "how much energy is required to complete a given computation?" Answering this question is nontrivial, in large part due to the complexity of superscalar architectures and the relationship between power and runtime. Good models to determine the energy cost prior to execution or models that are effective without experimentally collected application profiles were not discovered during the prepation of this survey paper. Several papers already discussed in this chapter looked at energy cost under different work placements; these works tend to note that different applications have different power consumption behaviors but do not attempt to explore the issue. The following papers pay some attention to this concern.

Work published in 2014 by Vogeleer et al. [15] reports a convex relationship between energy consumed and computation completed when varying the processor frequency. The authors spend a significant portion of the paper describing a model for energy loss that is compared against the experimental results. For the experiment reported, only a single benchmark is used. The experiment sweeps a space of frequency settings and input sizes. We observe from their results a consistent shape to the curve even though the computation per nanojoule can vary greatly based on input size.

Tiwari, Schulz, and Carrington [63] apply a machine learning approach to estimate the effect of a given power bound on a computation runtime. Roughly 10,000 compute kernel executions are collected to generate the training set for a machine learning algorithm. After training the model, the model is used to estimate application slowdown under a given bound. The authors experiment using only a single node and report an error of less than 6% between the experimentally measured and model estimated slowdown due to power bounding of two HPC benchmarks. Discussion of the similarities or differences in power consumption characteristics of the training compute kernels, benchmarks, and real HPC applications was not presented in the work and it is unclear if the models will continue to work in the general case or if the number of nodes is scaled.

Dynamic Power Sharing for Higher Job Throughput [19] includes discussion of an analytically devised model for simulation of application runtime under a power bound. In the derivation of this model, the author considers an application execution as an ordered sequence of transistor state transitions and an application to be modeled by a function of instantaneous power required. Modeling the application execution in this way allowed introduction of an Instruction Work metric, abstractly relating the instruction stream to the switching power. The runtimes reported by the model for execution under bound followed the experimentally observed trends. While not providing direct insight into the energy cost of a computation, an estimate of energy consumption using a model of this form could be produced through integration.

## 4.2 Power Aware Computing

For large HPC systems the power infrastructure is a major limitation on system size and in practice large HPC systems rarely run at the maximum power [49]. Patki et al. propose future HPC systems be hardware over-provisioned, meaning that more hardware is available than can be powered simultaneously at maximum consumption [48]. To safely operate such a system technologies such as RAPL are required to enforce that the aggregate power consumed does not exceed the system threshold [19].

Fukazawa et al. [22] look at power consumption of a single scientific application running under different RAPL power limits. Results from Fukazawa et al. indicate runtime increases nonlinearly beneath a limit threshold and that runtime does not improve from limits above the threshold. The MHD code studied is also interesting since there are distinct phases with different power consumption, the results indicate that power limits only impact the period of execution where an unlimited execution would consume above the limit. Ellsworth et al. replicate these result using several HPC benchmarks in Dynamic Power Sharing for Higher Job Throughput [19] and use the finding to drive the feedback mechanism for per socket power scheduling.

In Exploring Hardware Overprovisioning in Power-Constrained, High Performance Computing, Patki et al. [48] explore application performance under job wide power bound, enforced using RAPL. For the same power bound different node counts are possible; the job power bound, $L$, over $n$ nodes is satisfied by any node level power bound, $a$, such that $L \geq na$. Several HPC benchmark codes are used to evaluate the effect of bound and configuration on runtime performance. Patki et al. demonstrate that, under a job wide power bound, execution on the least number of nodes with the highest power setting never results in the shortest runtime and that the number of nodes and cores to achieve best performance is application specific.

In later work, Patki et al. [49] consider job scheduling in a hardware over-provisioned HPC system. Jobs are submitted with a processor count requirement and a power requirement. The job scheduler explored schedules jobs based on node availably and may reduce a job's power, by a configurable percent, if needed to fit the job within the available resources. The work demonstrates the ability to maintain a system wide power bound and improve job turn around time.

At the same conference, Ellsworth et al. [20] also present work showing the ability to enforce a power limit across an HPC system and reduce the time required to complete multiple concurrently executing jobs. The power enforcement system proposed by Ellsworth et al. monitors and opportunistically reallocates power between sockets at runtime. Sockets not consuming their allocation have the excess power allocation reclaimed by the scheduler and given to sockets consuming near their full allocation. An interesting note on the technique described is that no a priori knowledge of application power consumption characteristics or mapping of sockets to jobs is required. The work reports an observation of up to 14% runtime improvement, to complete all current jobs, over setting an equal power bound on all sockets for the duration of the experiment. Follow-on work by Ellsworth et al. [19] discusses the properties of the concurrently scheduled jobs needed to achieve runtime performance improvements. The power scheduling algorithm depends on differences between the power consumption across sockets. When all sockets would consume energy at the same rate there is no opportunity to yield power to other components within the system.

The discussion so far in has looked only at the power consumed by the processor. Many components contribute to the overall energy consumption and required power budget. Power costs for memory, network, and other devices must also be considered when looking at the total power consumption of an HPC system.

Tiwari, Schulz, and Carrington [63] look at the distribution between the CPU and DRAM within the system. The work takes advantage of the availability of RAPL on the experimental platform to set CPU and DRAM power limits independently. Prior to the experiment run, several computations are run at different power limits to generate training data. Training is done using training data and application features to produce a machine learned model capable of predicting the impact of bounds on an application.

Hoffmann frames the power and energy problems as a multidimensional optimization problem on program performance[27, 28]. Rather than looking at a bound as a hard limit, the works consider a bound as an optimization target when adjusting other properties of the environment. Environmental controls used in the work extend beyond direct power control, actions like changing the DVFS settings, to indirect controls, like changing the work distribution across cores. Hoffmann demonstrated an ability to reduce energy cost without performance impact for applications where iterations faster than a realtime interval produce no benefit [27]. Hoffmann et al. demonstrated a online tuning system, PTRADE, that can learn environmental settings to improve power efficiency during runtime [28].

## 4.3 Power Monitoring

Energy and power aware computing are currently topics of intense focus in the community due to the challenges associated with building ever larger HPC systems as well as the challenges in ubiquitous embedded computing. Many of the techniques for energy and power aware computing involve feedback mechanisms. An execution must be monitored in realtime and spooled for analysis and model generation or responded to directly for realtime execution guiding. The research systems mention earlier in this section implement monitoring, analysis, and control as tightly coupled software components. In addition to the direct research focus on how to best take advantage of the available energy there is also a need for efficient monitoring and analysis of energy and power attributes.

The IBM BlueGene systems include extensive power monitoring capabilities in the system management plane, via the BPM. Wallace et al. [65] show that the BPM measurements, while not at fine temporal granularity, can be useful for profiling applications however the measurements are not generally exposed to the jobs. The work by Wallace et al. also uses the environmental monitoring (EMON) API to capture similar data within a job and notes that, while EMON provides finer temporal resolution, the samples may not be tightly time aligned across components.

Hackenberg et al. [24] discuss work to improve the fidelity of energy measurement on their SandyBridge cluster. The authors note that accuracy can be lost at several points, starting with challenges in taking the analogue measurements and continuing through to the eventual storage. Measurements may also be taken at different levels within the machine and at different sampling intervals, complicating correlation between data streams. The work presents IPMI and BMC extensions that improve the accuracy and temporal resolution of power and energy measurements.

## 4.4 Takeaways

Power and energy are separate but related quantities. Energy consumption occurs at some rate during operation and power is a measure of the rate of energy consumption. Energy over an interval is the integral of instantaneous power over the interval. Conversion from energy to power over an interval produces an average power for the interval, which may not accurately represent the actual power if the interval is too large and the power changes. Examples of applications with phased power consumption indicate that allocating power based on dividing an applications running time by the run duration may negatively impact performance.

A tradeoff space exists between power, energy, and performance within the processor. Trivially a computer that is unplugged consumes no power but also fails to perform the desired computation. All interesting cases require some level of power consumption that supports computational progress. The energy used by a computation is the integral of power over the duration of the computation. Due to the design and physical properties of the processors, there is a direct relationship between instruction work per unit time (which is related to the clock speed) and power consumption. Reducing power consumption beneath a computation phase specific threshold extends the runtime of the computation, potentially increasing the total energy used. Increasing power beyond a computation phase specific threshold increases the total energy used and does not reduce runtime since NO-OP microinstructions must be executed to support operation at the higher clock frequency.

Assessment or estimation of the impact of changing a power setting would likely benefit from information across the system. Power control without visibility into application behavior may make poor decisions in the case of operation sequences, like busy waits, where lots of instructions are being executed but no application progress actually occurs. Some existing energy work in HPC uses the difference in arrival time at collectives to save power across nodes, increasing the temporal alignment and reducing power wasting wait time. Other work uses a notion of an application heartbeat to detect application progress as configuration changes are made.

# Chapter 5

# Conclusion and Perspectives

## 5.1 Recapitulation

Computing platforms are becoming increasingly limited by fiscal concerns rather than technical concerns. Large scale computing platforms carry significant capital and energy costs that must be justified by organizational management in terms of organizational objectives and constraints. 100 MW computers and 6 Ghz processors are technically possible to build however the costs to build and operate such systems are too high to justify in most cases. Viable computing platforms must balance management constraints, organizational objectives, and technical capabilities to receive funding for construction and ongoing operation.

Information Flow Processing (IFP) platforms potentially provide a mechanism to connect multiple management systems to the same data sources and provide some processing as the data flows toward the sinks. Intuitively IFP should provide value by amortizing data collection overheads across several sinks. Direct single solution performance comparison between IFP backed and non-IFP backed solutions may be unfair since there is no opportunity for overhead amortization. Advanced IFP platforms can provide the ability to process data as the data flows toward the sinks, allowing the processing to be distributed and reducing the required bandwidth at the end point.

Scheduling addresses how work and resources are matched in a shared environment. The problem can be approached as scheduling work onto resources, which typically is done by viewing scheduling as a queuing problem. While waiting in queue, the work is directly delayed by the scheduler. Alternatively the problem can be approached as scheduling resource to work, where resource requests are granted, failed, or some reduced resource allocation is provided. Work is fully serviced if the full allocation is granted or indirectly delayed if a lesser allocation is granted. Resource allocation and when work is executed impact the execution of organizational objectives. Ideally, schedulers will operate in alignment with organizational priorities and constraints.

Energy and power are important resources to manage that have specific qualities. Energy is consumed and converted to heat during computation. Power measures the rate of energy flow. Increasing energy efficiency does not necessarily reduce power and reducing power does not necessarily increase energy efficiency[1]. Reducing the processor clock speed, on existing processors, will reduce power consumption and may or may not impact application runtimes. High Performance Computing (HPC) systems are connected to the power grid and have access to effectively unbound energy but have a hard limit on the rate at which the energy can be accessed.

## 5.2 Perspective

Future computing platforms will need control and introspective capabilities to support the owning organization's objectives. The currently envisioned next generation HPC systems are far too large, too dynamic, and too fast to support human management of work and resources. Addressing this challenge will require introducing system management systems to dynamically control work and resource allocations within the platform. To make these decisions the system management system will need to be able to evaluate the system state during runtime and autonomously control the system to maintain alignment with organizational objectives. At this highest level, there are interesting considerations regarding how organizational intent can be expressed to the management system and how can system state be gathered and analyzed to check for fit with objectives.

Assuming the introspection problem can be solved, perhaps via an IFP-like mechanism, the results of the analysis must turn somehow into action. In most cases, one can imagine that a management action takes the form of a scheduling change. Both organization constraints and priorities would be reflected in the resource and work schedules executed by the platform. At this deeper level, there are interesting considerations regarding what information the schedulers need from

[1]Consider a component, $A$, that draws 15 watts over 3 seconds, for a total energy consumption of 45 joules. A more energy efficient component, $B$, might draw 20 watts over 2 seconds, for a total energy consumption of 40 joules. $B$ is more energy efficient than $A$ but also requires more power.

applications and the environment and what opportunities for performance improvement exist when schedulers and applications are aware of one another.

As a current hot topic in HPC research, power provides a concrete and focusing instance to explore the above questions and the interplay between management and technical system objectives. Power bound computing can be viewed as a problem of enforcing nontechnical constraints set by the owning organization. Enforcement of the bound may have an impact on the organizational objectives, as reduced power can reduce performance in terms of time to solution. Some mechanism must exist to distribute the information across subsystem boundaries for an enforcement mechanism to be aware of the impact of a change or for an application to be aware a change is coming. For a moment in time, there is an interesting opportunity to research at the intersection of IFP, scheduling, and power.

# Bibliography

[1] Top500. `http://www.top500.org/`. Accessed: 2016-02-18.

[2] Anthony Agelastos, Benjamin Allan, Jim Brandt, Paul Cassella, Jeremy Enos, Joshi Fullop, Ann Gentile, Steve Monk, Nichamon Naksinehaboon, Jeff Ogden, et al. The lightweight distributed metric service: a scalable infrastructure for continuous monitoring of large scale computing systems and applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 154–165. IEEE Press, 2014.

[3] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *NSDI*, volume 10, pages 19–19, 2010.

[4] Karim Ali and Ondřej Lhoták. Application-only call graph construction. In James Noble, editor, *ECOOP 2012 – Object-Oriented Programming*, volume 7313 of *Lecture Notes in Computer Science*, pages 688–712. Springer Berlin / Heidelberg, 2012.

[5] Arvind Arasu, Shivnath Babu, and Jennifer Widom. Cql: A language for continuous queries over streams and relations. In *Database Programming Languages*, pages 123–124. Springer, 2004.

[6] Dorian C Arnold and Barton P Miller. Scalable failure recovery for high-performance data aggregation. In *2010 IEEE International Symposium on Parallel&Distributed Processing (IPDPS)*.

[7] Mario Bambagini, Marko Bertogna, Mauro Marinoni, and Giorgio Buttazzo. An energy-aware algorithm exploiting limited preemptive scheduling under fixed priorities. In *Industrial Embedded Systems (SIES), 2013 8th IEEE International Symposium on*, pages 3–12. IEEE, 2013.

[8] Abhinav Bhatele, Kathryn Mohror, Steven H Langer, and Katherine E Isaacs. There goes the neighborhood: performance degradation due to nearby jobs. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 41. ACM, 2013.

[9] Alex D Breslow, Ananta Tiwari, Martin Schulz, Laura Carrington, Lingjia Tang, and Jason Mars. Enabling fair pricing on hpc systems with node sharing. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 37. ACM, 2013.

[10] Antonio Carzaniga, David S Rosenblum, and Alexander L Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems (TOCS)*, 19(3):332–383, 2001.

[11] Antonio Carzaniga and Alexander L Wolf. Forwarding in a content-based network. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 163–174. ACM, 2003.

[12] Mitch Cherniack, Hari Balakrishnan, Magdalena Balazinska, Donald Carney, Ugur Cetintemel, Ying Xing, and Stanley B Zdonik. Scalable distributed stream processing. In *CIDR*, volume 3, pages 257–268, 2003.

[13] Gianpaolo Cugola and Alessandro Margara. Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys (CSUR)*, 44(3):15, 2012.

[14] Jai Dayal, Dick Bratcher, Greg Eisenhauer, Karsten Schwan, Michael Wolf, Xuechen Zhang, Hasan Abbasi, Scott Klasky, and Norbert Podhorszki. Flexpath: Type-based publish/subscribe system for large-scale science analytics. In *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, pages 246–255. IEEE, 2014.

[15] Karel De Vogeleer, Gerard Memmi, Pierre Jouvelot, and Fabien Coelho. The energy/frequency convexity rule: Modeling and experimental validation on mobile devices. In *Parallel Processing and Applied Mathematics*, pages 793–803. Springer, 2014.

[16] Alan J Demers, Johannes Gehrke, Biswanath Panda, Mirek Riedewald, Varun Sharma, Walker M White, et al. Cayuga: A general purpose event monitoring system. In *CIDR*, volume 7, pages 412–422, 2007.

[17] Greg Eisenhauer, Karsten Schwan, and Fabián E Bustamante. Publish-subscribe for high-performance computing. *Internet Computing, IEEE*, 10(1):40–47, 2006.

[18] Greg Eisenhauer, Matthew Wolf, Hasan Abbasi, and Karsten Schwan. Event-based systems: opportunities and challenges at exascale. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, page 2. ACM, 2009.

[19] Daniel A Ellsworth, Allen D Malony, Barry Rountree, and Martin Schulz. Dynamic power sharing for higher job throughput. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 80. ACM, 2015.

[20] Daniel A. Ellsworth, Allen D. Malony, Barry Rountree, and Martin Schulz. Pow: System-wide dynamic reallocation of limited power in hpc. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '15, pages 145–148, New York, NY, USA, 2015. ACM.

[21] Dror G Feitelson, Larry Rudolph, and Uwe Schwiegelshohn. Parallel job schedulinga status report. In *Job Scheduling Strategies for Parallel Processing*, pages 1–16. Springer, 2005.

[22] Kyoko Fukazawa, Makoto Ueda, Masahiro Aoyagi, Tomonori Tsuhata, Kenta Yoshida, Aruta Uehara, Masakazu Kuze, Yuichi Inadomi, and Ken Inoue. Power consumption evaluation of an mhd simulation with cpu power capping. In *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, pages 612–617. IEEE, 2014.

[23] Rinku Gupta, Pete Beckman, B-H Park, Ewing Lusk, Paul Hargrove, Al Geist, Dhabaleswar K Panda, Andrew Lumsdaine, and Jack Dongarra. Cifts: A coordinated infrastructure for fault-tolerant systems. In *Parallel Processing, 2009. ICPP'09. International Conference on*, pages 237–245. IEEE, 2009.

[24] Daniel Hackenberg, Thomas Ilsche, Joseph Schuchart, Robert Schone, Wolfgang E Nagel, Marc Simon, and Yiannis Georgiou. Hdeem: high definition energy efficiency monitoring. In *Energy Efficient Supercomputing Workshop (E2SC), 2014*, pages 1–10. IEEE, 2014.

[25] Martin Hirzel, Robert Soulé, Scott Schneider, Buğra Gedik, and Robert Grimm. A catalog of stream processing optimizations. *ACM Computing Surveys (CSUR)*, 46(4):46, 2014.

[26] Torsten Hoefler and Timo Schneider. Optimization principles for collective neighborhood communications. In *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, pages 1–10. IEEE, 2012.

[27] Henry Hoffmann. Racing and pacing to idle: an evaluation of heuristics for energy-aware resource allocation. In *Proceedings of the Workshop on Power-Aware Computing and Systems*, page 13. ACM, 2013.

[28] Henry Hoffmann, Martina Maggio, Marco D Santambrogio, Alberto Leva, and Abhishek Agarwal. A generalized software framework for accurate and efficient management of performance goals. In *Embedded Software (EMSOFT), 2013 Proceedings of the International Conference on*, pages 1–10. IEEE, 2013.

[29] Yuichi Inadomi, Tapasya Patki, Koji Inoue, Mutsumi Aoyagi, Barry Rountree, Martin Schulz, David Lowenthal, Yasutaka Wada, Keiichiro Fukazawa, Masatsugu Ueda, et al. Analyzing and mitigating the impact of manufacturing variability in power-constrained supercomputing. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 78. ACM, 2015.

[30] Hans-Arno Jacobsen, Alex King Yeung Cheung, Guoli Li, Balasubramaneyam Maniymaran, Vinod Muthusamy, and Reza Sherafat Kazemzadeh. The padres publish/subscribe system. *Principles and Applications of Distributed Event-Based Systems*, 164:205, 2010.

[31] Terry Jones and Gregory A Koenig. A clock synchronization strategy for minimizing clock variance at runtime in high-end computing environments. In *Computer Architecture and High Performance Computing (SBAC-PAD), 2010 22nd International Symposium on*, pages 207–214. IEEE, 2010.

[32] Gueyoung Jung, Matti A Hiltunen, Kaustubh R Joshi, Richard D Schlichting, and Calton Pu. Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures. In *Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference on*, pages 62–73. IEEE, 2010.

[33] Melanie Kambadur, Tipp Moseley, Rick Hank, and Martha A Kim. Measuring interference between live datacenter applications. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 51. IEEE Computer Society Press, 2012.

[34] Alexandros Koliousis and Joseph Sventek. Glasgow automata illustrated: Debs grand challenge. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, pages 353–358. ACM, 2012.

[35] Athanasios Konstantinidis, Antonio Carzaniga, and Alexander L Wolf. A content-based publish/subscribe matching algorithm for 2d spatial objects. In *Proceedings of the 12th International Middleware Conference*, pages 200–219. International Federation for Information Processing, 2011.

[36] Mahendra Kutare, Greg Eisenhauer, Chengwei Wang, Karsten Schwan, Vanish Talwar, and Matthew Wolf. Monalytics: online monitoring and analytics for managing large scale data centers. In *Proceedings of the 7th international conference on Autonomic computing*, pages 141–150. ACM, 2010.

[37] Monica S Lam, John Whaley, V Benjamin Livshits, Michael C Martin, Dzintars Avots, Michael Carbin, and Christopher Unkel. Context-sensitive program analysis as database queries. In *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 1–12. ACM, 2005.

[38] JLJ Laredo, P Bouvry, F Guinand, B Dorronsoro, and C Fernandes. The sandpile scheduler. *Cluster Computing*, 17(2):191–204, 2014.

[39] Gregory L Lee, Dong H Ahn, Dorian C Arnold, Bronis R De Supinski, Matthew Legendre, Barton P Miller, Martin Schulz, and Ben Liblit. Lessons learned at 208k: towards debugging millions of cores. In *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for*, pages 1–9. IEEE, 2008.

[40] Guoli Li and Hans-Arno Jacobsen. Composite subscriptions in content-based publish/subscribe systems. In *Middleware 2005*, pages 249–269. Springer, 2005.

[41] Mark A Linton. Implementing relational views of programs. In *ACM SIGSOFT Software Engineering Notes*, volume 9, pages 132–140. ACM, 1984.

[42] Shengjian Liu, Yongheng Wang, Shuguang Peng, and Xinlong Zhang. High efficient complex event processing based on storm. In *The Proceedings of the Second International Conference on Communications, Signal Processing, and Systems*, pages 671–678. Springer, 2014.

[43] Jay F. Lofstead, Scott Klasky, Karsten Schwan, Norbert Podhorszki, and Chen Jin. Flexible io and integration for scientific codes through the adaptable io system (adios). In *Proceedings of the 6th International Workshop on Challenges of Large Applications in Distributed Environments*, CLADE '08, pages 15–24, New York, NY, USA, 2008. ACM.

[44] Kazuhiro Minami, Adam J Lee, Marianne Winslett, and Nikita Borisov. Secure aggregation in a publish-subscribe system. In *Proceedings of the 7th ACM workshop on Privacy in the electronic society*, pages 95–104. ACM, 2008.

[45] Robert J Minerick, Vincent W Freeh, and Peter M Kogge. Dynamic power management using feedback. 2002.

[46] Aroon Nataraj, Allen D Malony, Alan Morris, Dorian C Arnold, and Barton P Miller. A framework for scalable, parallel performance monitoring. *Concurrency and Computation: Practice and Experience*, 22(6):720–735, 2010.

[47] Kay Ousterhout, Patrick Wendell, Matei Zaharia, and Ion Stoica. Sparrow: distributed, low latency scheduling. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 69–84. ACM, 2013.

[48] Tapasya Patki, David K. Lowenthal, Barry Rountree, Martin Schulz, and Bronis R. de Supinski. Exploring hardware overprovisioning in power-constrained, high performance computing. In *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*, ICS '13, pages 173–182, New York, NY, USA, 2013. ACM.

[49] Tapasya Patki, David K. Lowenthal, Anjana Sasidharan, Matthias Maiterth, Barry L. Rountree, Martin Schulz, and Bronis R. de Supinski. Practical

resource management in power-constrained, high performance computing. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '15, pages 121–132, New York, NY, USA, 2015. ACM.

[50] Olga Pearce, Todd Gamblin, Bronis R. de Supinski, Tom Arsenlis, and Nancy M. Amato. Load balancing n-body simulations with highly non-uniform density. In *Proceedings of the 28th ACM International Conference on Supercomputing*, ICS '14, pages 113–122, New York, NY, USA, 2014. ACM.

[51] Olga Pearce, Todd Gamblin, Bronis R de Supinski, Martin Schulz, and Nancy M Amato. Quantifying the effectiveness of load balance algorithms. In *Proceedings of the 26th ACM international conference on Supercomputing*, pages 185–194. ACM, 2012.

[52] Olga Pearce, Todd Gamblin, Bronis R. de Supinski, Martin Schulz, and Nancy M. Amato. Decoupled load balancing. In *Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP 2015, pages 267–268, New York, NY, USA, 2015. ACM.

[53] Swann Perarnau, Rajeev Thakur, Kamil Iskra, Ken Raffenetti, Franck Cappello, Rinku Gupta, Pete Beckman, Marc Snir, Henry Hoffmann, Martin Schulz, et al. Distributed monitoring and management of exascale systems in the argo project. In *Distributed Applications and Interoperable Systems*, pages 173–178. Springer, 2015.

[54] Peter R Pietzuch and Jean M Bacon. Hermes: A distributed event-based middleware architecture. In *Distributed Computing Systems Workshops, 2002. Proceedings. 22nd International Conference on*, pages 611–618. IEEE, 2002.

[55] Philip C Roth, Dorian C Arnold, and Barton P Miller. Mrnet: A software-based multicast/reduction network for scalable tools. In *Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, page 21. ACM, 2003.

[56] Barry Rountree, Dong H. Ahn, Bronis R. de Supinski, David K. Lowenthal, and Martin Schulz. Beyond dvfs: A first look at performance under a hardware-enforced power bound. In *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, IPDPSW '12, pages 947–953, Washington, DC, USA, 2012. IEEE Computer Society.

[57] Barry Rountree, David K Lownenthal, Bronis R de Supinski, Martin Schulz, Vincent W Freeh, and Tyler Bletsch. Adagio: making dvs practical for complex hpc applications. In *Proceedings of the 23rd international conference on Supercomputing*, pages 460–469. ACM, 2009.

[58] Sushant Sharma, Dimitrios Katramatos, Dantong Yu, and Li Shi. Design and implementation of an intelligent end-to-end network qos system. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 68. IEEE Computer Society Press, 2012.

[59] Yannis Smaragdakis and Martin Bravenboer. Using datalog for fast and easy program analysis. In Oege de Moor, Georg Gottlob, Tim Furche, and Andrew Sellers, editors, *Datalog Reloaded*, volume 6702 of *Lecture Notes in Computer Science*, pages 245–251. Springer Berlin / Heidelberg, 2011.

[60] Michael Stonebraker, Uur Çetintemel, and Stan Zdonik. The 8 requirements of real-time stream processing. *ACM SIGMOD Record*, 34(4):42–47, 2005.

[61] Joseph Sventek and Alexandros Koliousis. Unification of publish/subscribe systems and stream databases: the impact on complex event processing. In *Proceedings of the 13th International Middleware Conference*, pages 292–311. Springer-Verlag New York, Inc., 2012.

[62] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. Optimization of collective communication operations in mpich. *International Journal of High Performance Computing Applications*, 19(1):49–66, 2005.

[63] Ananta Tiwari, Martin Schulz, and Laura Carrington. Predicting optimal power allocation for cpu and dram domains. In *Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International*, pages 951–959. IEEE, 2015.

[64] Anish Tiwari, Michael Laurenzano, Joshua Peraza, Laura Carrington, and Allan Snavely. Green queue: Customized large-scale clock frequency scaling. In *Cloud and Green Computing (CGC), 2012 Second International Conference on*, pages 260–267. IEEE, 2012.

[65] Sean Wallace, Venkatram Vishwanath, Susan Coghlan, Zhiling Lan, and Michael E Papka. Measuring power consumption on ibm blue gene/q. In

*Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International*, pages 853–859. IEEE, 2013.

[66] Chengwei Wang, Karsten Schwan, Vanish Talwar, Greg Eisenhauer, Liting Hu, and Matthew Wolf. A flexible architecture integrating monitoring and analytics for managing large-scale data centers. In *Proceedings of the 8th ACM international conference on Autonomic computing*, pages 141–150. ACM, 2011.

[67] Eugene Wu, Yanlei Diao, and Shariq Rizvi. High-performance complex event processing over streams. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 407–418. ACM, 2006.

[68] Xing Xie, Indrakshi Ray, Raman Adaikkalavan, and Rose Gamble. Information flow control for stream processing in clouds. In *Proceedings of the 18th ACM symposium on Access control models and technologies*, pages 89–100. ACM, 2013.

[69] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.

[70] Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, and Ion Stoica. Discretized streams: Fault-tolerant streaming computation at scale. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 423–438. ACM, 2013.

[71] Gengbin Zheng, Abhinav Bhatele, Esteban Meneses, and Laxmikant V Kalé. Periodic hierarchical load balancing for large supercomputers. *International Journal of High Performance Computing Applications*, page 1094342010394383, 2011.

# Works Not Discussed

[72] Hasan Abbasi, Matthew Wolf, Greg Eisenhauer, Scott Klasky, Karsten Schwan, and Fang Zheng. Datastager: scalable data staging services for petascale applications. In *Proceedings of the 18th ACM international symposium on High performance distributed computing*, pages 39–48. ACM, 2009.

[73] Naveed Arshad, Dennis Heimbigner, and Alexander L Wolf. Deployment and dynamic reconfiguration planning for distributed software systems. In *Tools with Artificial Intelligence, 2003. Proceedings. 15th IEEE International Conference on*, pages 39–46. IEEE, 2003.

[74] Roberto Baldoni and Antonino Virgillito. Distributed event routing in publish/subscribe communication systems: a survey. *DIS, Universita di Roma La Sapienza, Tech. Rep*, page 5, 2005.

[75] Abhinav Bhatele, Todd Gamblin, Katherine E Isaacs, Brian TN Gunney, Martin Schulz, Peer-Timo Bremer, and Bernd Hamann. Novel views of performance data to analyze large-scale adaptive applications. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 31. IEEE Computer Society Press, 2012.

[76] Huy Bui, Hal Finkel, Venkatram Vishwanath, Salman Habib, Katrin Heitmann, Jason Leigh, Michael Papka, and Kevin Harms. Scalable parallel i/o on a blue gene/q supercomputer using compression, topology-aware data aggregation, and subfiling. In *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on*, pages 107–111. IEEE, 2014.

[77] Mauro Caporuscio, Antonio Carzaniga, and Alexander Wolf. An experience in evaluating publish/subscribe services in a wireless network. In *Proceedings of the 3rd international workshop on Software and performance*, pages 128–133. ACM, 2002.

[78] Raphaël Chand and Pascal Felber. Semantic peer-to-peer overlays for publish/subscribe networks. In *Euro-Par 2005 Parallel Processing*, pages 1194–1204. Springer, 2005.

[79] Tiancheng Chang, Sisi Duan, Hein Meling, Sean Peisert, and Haibin Zhang. P2s: a fault-tolerant publish/subscribe infrastructure. In *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, pages 189–197. ACM, 2014.

[80] Gianpaolo Cugola and Alessandro Margara. High-performance location-aware publish-subscribe on gpus. In *Middleware 2012*, pages 312–331. Springer, 2012.

[81] John Daly, B Harrod, T Hoang, L Nowell, B Adolf, S Borkar, N DeBardeleben, M Elnozahy, M Heroux, D Rogers, et al. Inter-agency workshop on hpc resilience at extreme scale. *National Security Agency Advanced Computing Systems*, 2012.

[82] Christian Engelmann and Thomas Naughton. Toward a performance/resilience tool for hardware/software co-design of high-performance computing systems. In *Parallel Processing (ICPP), 2013 42nd International Conference on*, pages 960–969. IEEE, 2013.

[83] Hadi Esmaeilzadeh, Emily Blem, Renee St Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*, pages 365–376. IEEE, 2011.

[84] Patrick Eugster. Type-based publish/subscribe: Concepts and experiences. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 29(1):6, 2007.

[85] Patrick Eugster and KR Jayaram. Eventjava: An extension of java for event correlation. In *ECOOP 2009–Object-Oriented Programming*, pages 570–594. Springer, 2009.

[86] Patrick Th Eugster, Pascal A Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys (CSUR)*, 35(2):114–131, 2003.

[87] Patrick Th Eugster, Rachid Guerraoui, and Joe Sventek. Distributed asynchronous collections: Abstractions for publish/subscribe interaction. In *ECOOP 2000Object-Oriented Programming*, pages 252–276. Springer, 2000.

[88] Thomas Fischer, Andreas M Wahl, and Richard Lenz. Automated quality-of-service-aware configuration of publish-subscribe systems at design-time. In *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, pages 118–129. ACM, 2014.

[89] Michael J Franklin, Shawn R Jeffery, Sailesh Krishnamurthy, Frederick Reiss, Shariq Rizvi, Eugene Wu, Owen Cooper, Anil Edakkunni, and Wei Hong. Design considerations for high fan-in systems: The hifi approach. In *CIDR*, volume 5, pages 24–27, 2005.

[90] Al Geist, Shekhar Borkar, Eric Roman, Bert Still, Robert Clay SNL, John Wu, Christian Engelmann, Nathan DeBardeleben, Larry Kaplan, Martin Schulz, et al. Us department of energy fault management workshop. In *Workshop report submitted to the US Department of Energy*, 2012.

[91] Aaron Gember-Jacobson, Raajay Viswanathan, Chaithan Prakash, Robert Grandl, Junaid Khalid, Sourav Das, and Aditya Akella. Opennf: Enabling innovation in network function control. *ACM SIGCOMM Computer Communication Review*, 44(4):163–174, 2015.

[92] Akram Hakiri, Pascal Berthou, Aniruddha Gokhale, Douglas C Schmidt, and Thierry Gayraud. Supporting end-to-end quality of service properties in omg data distribution service publish/subscribe middleware over wide area networks. *Journal of Systems and Software*, 86(10):2574–2593, 2013.

[93] Thomas Heinze, Zbigniew Jerzak, Gregor Hackenbroich, and Christof Fetzer. Latency-aware elastic scaling for distributed data stream processing systems. In *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, pages 13–22. ACM, 2014.

[94] Matthias Hovestadt, Odej Kao, Axel Keller, and Achim Streit. Scheduling in hpc resource management systems: Queuing vs. planning. In *Job Scheduling Strategies for Parallel Processing*, pages 1–20. Springer, 2003.

[95] Zbigniew Jerzak and Christof Fetzer. Bloom filter based routing for content-based publish/subscribe. In *Proceedings of the second international conference on Distributed event-based systems*, pages 71–81. ACM, 2008.

[96] Qingchun Jiang and Sharma Chakravarthy. Scheduling strategies for processing continuous queries over streams. In *Key Technologies for Data Management*, pages 16–30. Springer, 2004.

[97] Petri Jokela, András Zahemszky, Christian Esteve Rothenberg, Somaya Arianfar, and Pekka Nikander. Lipsin: line speed publish/subscribe inter-networking. *ACM SIGCOMM Computer Communication Review*, 39(4):195–206, 2009.

[98] Reza Sherafat Kazemzadeh and Hans-Arno Jacobsen. Reliable and highly available distributed publish/subscribe service. In *Reliable Distributed Systems, 2009. SRDS'09. 28th IEEE International Symposium on*, pages 41–50. IEEE, 2009.

[99] James F. Kurose and W. Keith Ross. *Computer networking: a top-down approach featuring the Internet.* Addison-Wesley, 2003.

[100] Ignacio Laguna, Edgar A León, Martin Schulz, and Mark Stephenson. A study of application-level recovery methods for transient network faults. In *Proceedings of the Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*, page 3. ACM, 2013.

[101] Matteo Migliavacca and Gianpaolo Cugola. Adapting publish-subscribe routing to traffic demands. In *Proceedings of the 2007 inaugural international conference on Distributed event-based systems*, pages 91–96. ACM, 2007.

[102] Andrew C Myers and Barbara Liskov. *A decentralized model for information flow control*, volume 31. ACM, 1997.

[103] Rimma V Nehme, Hyo-Sang Lim, and Elisa Bertino. Fence: Continuous access control enforcement in dynamic data stream environments. In *Proceedings of the third ACM conference on Data and application security and privacy*, pages 243–254. ACM, 2013.

[104] Xiangyong Ouyang, Raghunath Rajachandrasekar, Xavier Besseron, and Dhabaleswar K Panda. High performance pipelined process migration with

rdma. In *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 314–323. IEEE Computer Society, 2011.

[105] Navneet Kumar Pandey, Kaiwen Zhang, Stéphane Weiss, Hans-Arno Jacobsen, and Roman Vitenberg. Distributed event aggregation for content-based publish/subscribe systems. In *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, pages 95–106. ACM, 2014.

[106] Adrian Paschke, Paul Vincent, Alex Alves, and Catherine Moxey. Tutorial on advanced design patterns in event processing. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, pages 324–334. ACM, 2012.

[107] Jacques A Pienaar, Srimat Chakradhar, and Anand Raghunathan. Automatic generation of software pipelines for heterogeneous parallel systems. In *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, pages 1–12. IEEE, 2012.

[108] Ankit Singla, Chi-Yao Hong, Lucian Popa, and P Brighten Godfrey. Jellyfish: Networking data centers randomly. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 225–238, 2012.

[109] John A Stankovic, Tian He, Tarek Abdelzaher, Mike Marley, Gang Tao, Sang Son, and Chenyang Lu. Feedback control scheduling in distributed real-time systems. In *Real-Time Systems Symposium, 2001.(RTSS 2001). Proceedings. 22nd IEEE*, pages 59–70. IEEE, 2001.

[110] Kevin Tomsovic, David E Bakken, Vaithianathan Venkatasubramanian, and Anjan Bose. Designing the next generation of real-time control, communication, and computations for large power systems. *Proceedings of the IEEE*, 93(5):965–979, 2005.

[111] Christos Tsilopoulos, Ioannis Gasparis, George Xylomenos, and George C Polyzos. Efficient real-time information delivery in future internet publish-subscribe networks. In *Computing, Networking and Communications (ICNC), 2013 International Conference on*, pages 856–860. IEEE, 2013.

[112] Venkatram Vishwanath, Mark Hereld, Vitali Morozov, and Michael E Papka. Topology-aware data movement and staging for i/o acceleration on blue gene/p supercomputing systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 19. ACM, 2011.

[113] Ke Wang, Kevin Brandstatter, and Ioan Raicu. Simmatrix: Simulator for many-task computing execution fabric at exascale. In *Proceedings of the High Performance Computing Symposium*, page 9. Society for Computer Simulation International, 2013.

[114] Bing Xie, Jeffrey Chase, David Dillow, Oleg Drokin, Scott Klasky, Sarp Oral, and Norbert Podhorszki. Characterizing output bottlenecks in a supercomputer. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 8. IEEE Computer Society Press, 2012.

[115] Xin Yuan, Santosh Mahapatra, Wickus Nienaber, Scott Pakin, and Michael Lang. A new routing scheme for jellyfish and its performance with hpc workloads. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 36. ACM, 2013.

[116] Ye Zhao, Kyungbaek Kim, and Nalini Venkatasubramanian. Dynatops: A dynamic topic-based publish/subscribe architecture. In *Proceedings of the 7th ACM international conference on Distributed event-based systems*, pages 75–86. ACM, 2013.